DUY TÂN UNIVERSITY

International School

Capstone Project 1

CMU-SE 450

Architecture Design
Version 1.1
Date: 20 December 2024

ResumeGeniusAI - AI-Powered Resume Builder & Analysis integrated with the Job Search Platform

Submitted by C1SE.02

Vo Van Minh
Doan Ngoc Dat
Tran Duong Truong
Ta Dinh Tai

Approved by Tran Kim Sanh

Architecture Review Panel Representative:

Name Signature Date

Capstone Project 1- Mentor:

Name Signature Date

Project Information

Project acronym	RGA		
Project Title	AI-Powered Resume Builder & Analysis integrated with the Job Search Platform		
Start Date	26 Sep 2024	**End Date**	20 Dec 2024
Lead Institution	International School, Duy Tan University		
Project Mentor	Tran Kim Sanh, MSc Email: sanhtk@gmail.com Tel: 0987409464		
Scrum Master & contact details	Vo Van Minh Email: vovanminhv23@gmail.com Tel: 0917482032		
Partner Organization			
Team members	Name	Email	Tel
27211245990	Ta Dinh Tai	tadinhtai6868@gmail.com	0373444993
27211239967	Tran Duong Truong	tranduongtruong1623@gmail.com	0934970854
27211245795	Doan Ngoc Dat	datngoc252k3@gmail.com	0856794061

DOCUMENT INFORMATION

Document Title	Architecture Design		
Author(s)	Team C1SE.02		
Role	**Name**		
Scrum Master	Vo Van Minh		
Product Owner	Tran Duong Truong		
Developer	Doan Ngoc Dat, Ta Dinh Tai		
Date	20 Dec 2024	**File name**	C1SE.02_ Architecture_Design_RGA(v1.1).docx
URL			
Access	Project and CMU Program		

REVISION HISTORY

Version	Date	Comments	Author	Approval
1.0	15 Nov 2024	Internal Document	Group C1SE.02	
1.1	12 Dec 2024	Format Document	Doan Ngoc Dat	

SIGNATURE

Document Approvals: The following signatures are required for approval of this document.

Tran Kim Sanh, MSc *Mentor*	Signature	Date
Vo Van Minh Student ID: 27211226524 *Scrum Master*	Signature	Date

CONTENTS

Table

Figure

1. Introduction

1.1. Purpose

The purpose of the Architecture document is to:

- Define the architecture needs and technology in detail.
- Provide solutions for business needs.
- Provide an overview of resources, schedule, solution, and budget for the project.

The architecture merely introduces the project to the student development teams and provides the up-front information necessary for the team to develop a specification.

1.2. Documents references

Table 1.1 *Table about documents references.*

No.	Reference
1	Product Backlog Document v1.1
2	Project Plan Document v1.1
3	Database Design Document v1.1

2. Project Statement

2.1. Project Definition

The Resume Builder Website project aims to provide a comprehensive platform for Candidates, HR professionals, and Administrators to efficiently create, manage, and share resumes. The website offers features such as resume creation with customizable templates, automatic upgrades for existing resumes, job posting and application management, and personalized job notifications. Leveraging advanced AI capabilities, the platform enhances resumes with grammar correction, sentence optimization, and automatic summary generation. The backend uses a combination of NestJS and Flask with MongoDB for data storage, offering a seamless user experience.

2.2. Business needs/ User needs

In today’s competitive job market, candidates are looking for efficient ways to stand out, and one of the most crucial aspects is having a professional, well-crafted resume. However, many candidates struggle with creating high-quality resumes that

effectively highlight their skills and experiences. They need a platform that provides easy-to-use tools for resume creation, with features such as customizable templates, automatic grammar corrections, and content optimization. Additionally, with the rapid advancement of artificial intelligence, candidates are increasingly seeking AI-powered features that can automatically enhance their resumes, improving readability and making them more appealing to employers.

On the other hand, HR professionals face the challenge of managing large volumes of applications, often struggling to find the right candidates quickly. They need a system that allows them to post job listings, efficiently track applicants, and view resumes that are professionally optimized. Furthermore, both candidates and HR professionals need an automated notification system that keeps them informed about new job opportunities or applications, saving time and improving the hiring process. This project aims to address these needs by providing a seamless, AI-powered platform that helps candidates create optimized resumes while simplifying the recruitment process for HR professionals.

2.3. Project Goal

The project aims to build an easy-to-use platform that empowers candidates to create and optimize their resumes with AI-powered features. It will integrate AI tools to enhance resumes by correcting grammar, improving sentence structure, and auto-generating content such as summaries and job descriptions. The platform will provide HR with an efficient way to post jobs, track applications, and receive real-time notifications when candidates apply. Additionally, the system will send automated email notifications to candidates about new job postings and weekly updates, ensuring seamless communication. The goal is to deliver a secure, scalable, and user-friendly platform for both candidates and employers.

3. Architecture Drivers

3.1. High-level requirements

(Refer to the Product Backlog document for RGA)

3.2. System Overview

3.2.1. System Context

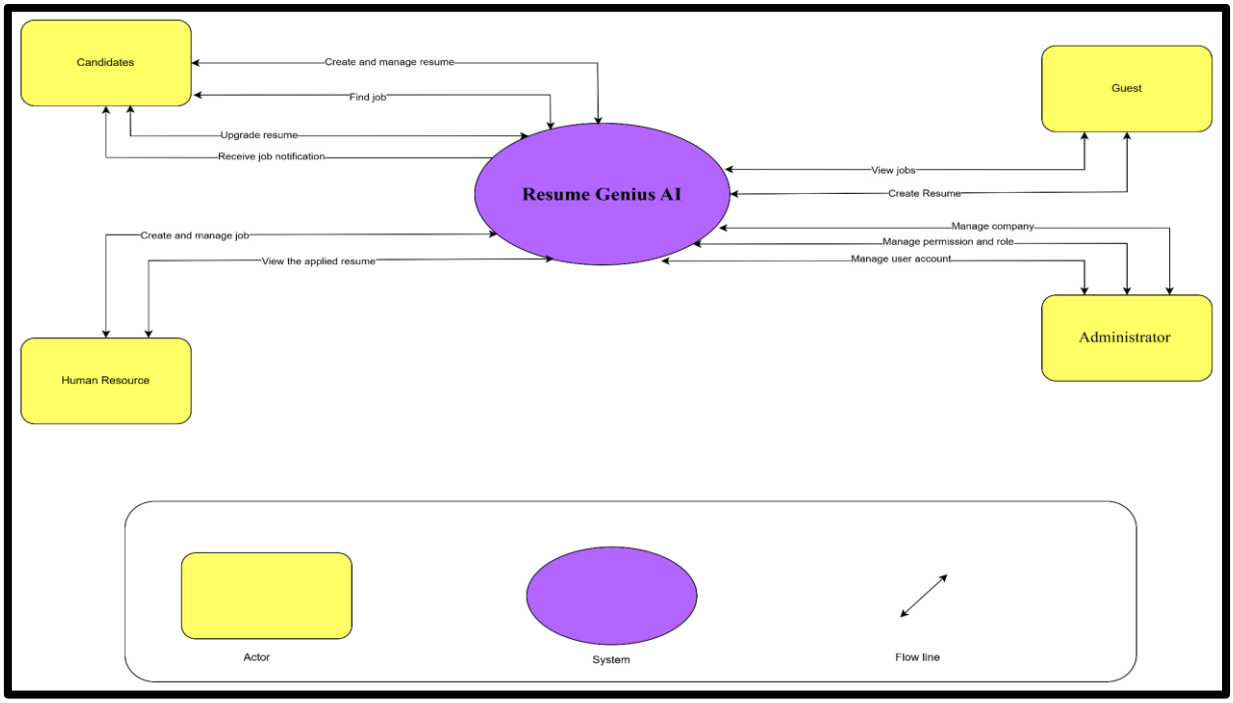

Figure 3.1 *System context overview.*

3.2.2. System Context description

The diagram shows the following flows of information:

Candidate register and log into the system, can create a resume using pre-existing templates or upload their own resumes for analysis and enhancement with an AI power tool. They can also search for job openings and apply for positions directly through the platform.

HR professionals register and log in. They can post job descriptions, search for suitable candidates, and communicate with job seekers via email. They can manage job postings and interact with applicants to find the best fit for their vacancies.

Admins manage user accounts, oversee resume submissions, and job descriptions, and ensure the smooth operation of all system functionalities. They handle user permissions, content management, and system maintenance.

Guests can view job listings and create resumes. However, they cannot apply for jobs or save the created resume until they register and log in as a Candidate or HR.

3.3. Quality Attributes

Table 3.1 *Table about quality attributes (QA01)*

ID	**QA01**
Title	User-Friendly Interface
Quality Attribute	Usability
Stakeholder Role	User (Candidates, HR, Admin, Guests)
Source(s) of Stimulus	User interaction with the system
Stimulus	The user attempts to navigate the interface and perform tasks
Relevant Environment Conditions	Normal operational conditions
Architectural Elements	User Interface, Menu Browser, Resume Builder, Job Application.
System Response	The system should allow users to navigate seamlessly, and perform tasks such as creating resumes, applying for jobs, and communicating with HR without confusion or errors.
Response Measure(s)	Task completion time, user satisfaction ratings, error rate
Associated Risks	High task completion time, low user satisfaction, frequent user errors

Table 3.2 *Table about quality attributes (QA02)*

ID	**QA02**
Title	Data Security
Quality Attribute	Security
Stakeholder Role	All users (Candidates, HR, Admin, Guests)
Source(s) of Stimulus	Malicious actors, unauthorized users

Stimulus	Attempt to access the system or data without proper authorization
Relevant Environment Conditions	The system is online and accessible over the internet
Architectural Elements	Authentication module, authorization module, encryption mechanisms, firewall
System Response	Deny access, log the attempt, alert admin
Response Measure(s)	Number of unauthorized access attempts detected and prevented, response time to security incidents
Associated Risks	Data breaches, compromised user accounts, legal and regulatory penalties

Table 3.3 *Table about quality attributes (QA03)*

ID	**QA03**
Title	High Performance
Quality Attribute	Performance
Stakeholder Role	Candidates, HR, Admin
Source(s) of Stimulus	User actions (e.g., uploading a resume, searching for jobs)
Stimulus	The high volume of user requests, peak usage times
Relevant Environment Conditions	Normal operation with a high number of concurrent users
Architectural Elements	Load balancer, optimized database queries, efficient algorithms, caching mechanisms
System Response	Process user requests in a timely manner
Response Measure(s)	System response time, throughput, number of concurrent users supported
Associated Risks	Slow system performance, user dissatisfaction, loss of users

Table 3.4 *Table about quality attributes (QA04)*

ID	**QA04**
Title	Scalability
Quality Attribute	Scalability
Stakeholder Role	Candidates, HR, Admin
Source(s) of Stimulus	Increase in user base, increase in data volume
Stimulus	High growth in the number of users and resumes processed
Relevant Environment Conditions	Growing user base, increased job postings and applications
Architectural Elements	Scalable database architecture, microservices architecture, cloud infrastructure
System Response	Scale up or down based on demand
Response Measure(s)	Number of users supported, system performance under load, resource utilization
Associated Risks	Resource exhaustion, degraded performance, higher operational costs

4. Constraints

4.1. Business Constraints

Table 4.1 *Table about business constraints (BC01)*

ID	**BC01**
Title	Budget Limitation
Description	The project has a budget of $10,000. Any additional expenses beyond this budget must be justified and approved by the project stakeholders.

Table 4.2 *Table about business constraints (BC02)*

ID	**BC02**
Title	Resource Availability
Description	The project team consists of 4 people. Any changes or additions to the team must be approved by the project manager.

Table 4.3 *Table about business constraints (BC03)*

ID	**BC03**
Title	Time Constraint
Description	The project must be completed within 4 months from the project initiation date. Any delays or need for timeline adjustments must be documented and approved by the project management team.

Table 4.4 *Table about business constraints (BC04)*

ID	**BC04**
Title	Quality Standards
Description	All project deliverables must align with the quality benchmarks specified in the project documentation. Continuous quality assessments and reviews will be conducted to ensure compliance throughout the project duration.

Table 4.5 *Table about business constraints (BC05)*

ID	**BC05**
Title	Scope Definition
Description	The project scope should be clearly outlined and documented from the outset. Any modifications to the scope must be formally approved by the project sponsor to prevent unauthorized scope changes.

Table 4.6 *Table about business constraints (BC06)*

ID	**BC06**
Title	Communication Plan
Description	A well-defined communication plan must be established and adhered to throughout the project. Frequent status updates and relevant project information should be communicated to all stakeholders on time.

Table 4.7 *Table about business constraints (BC07)*

ID	**BC07**
Title	Risk Management
Description	A robust risk management plan is required to proactively identify, assess, and mitigate potential risks throughout the project's lifecycle. Any critical risks should be reported and addressed promptly.

Table 4.8 *Table about business constraints (BC08)*

ID	**BC08**
Title	Stakeholder Engagement
Description	Ongoing communication and engagement with project stakeholders is crucial. A strategy should be established to identify stakeholders, address their concerns, and meet their needs during the project.

Table 4.9 *Table about business constraints (BC09)*

ID	**BC09**
Title	Legal Compliance
Description	The project must comply with all applicable legal and regulatory standards. Regular audits and documentation must be conducted to ensure full compliance throughout the project's progression.

4.2. Technical Constraints

Table 4.10 *Table about technical constraints (TC01)*

ID	**TC01**
Title	Backend Framework
Description	Main Programming Language: JavaScript/TypeScript, Python Backend Framework: NestJS (for general backend services) AI Frameworks: Python Flask (for AI-related services) Database: MongoDB Development Tools: Visual Studio Code, PyCharm Containerization: Docker (for microservices deployment)

Table 4.11 *Table about business constraints (BC02)*

ID	**TC02**
Title	Browser Compatibility
Description	The frontend application must support the following browsers: Chrome, Firefox, Safari, Edge Ensure consistent performance and appearance across all supported browsers

Table 4.12 *Table about business constraints (BC03)*

ID	**TC03**
Title	Frontend Framework
Description	Main Programming Language: JavaScript/TypeScript Framework: React CSS Framework: TailwindCSS Build Tools: Webpack, Vite Development Environment: Visual Studio Code

Table 4.13 *Table about business constraints (BC04)*

ID	**TC04**
Title	Server Hosting Requirements
Description	CPU: 2 vCPU Memory: 8GB RAM Storage: 40GB SSD Web Server: Netlify Application Server: Render

5. High-Level Architecture

5.1. Component and Connector view (C&C view)

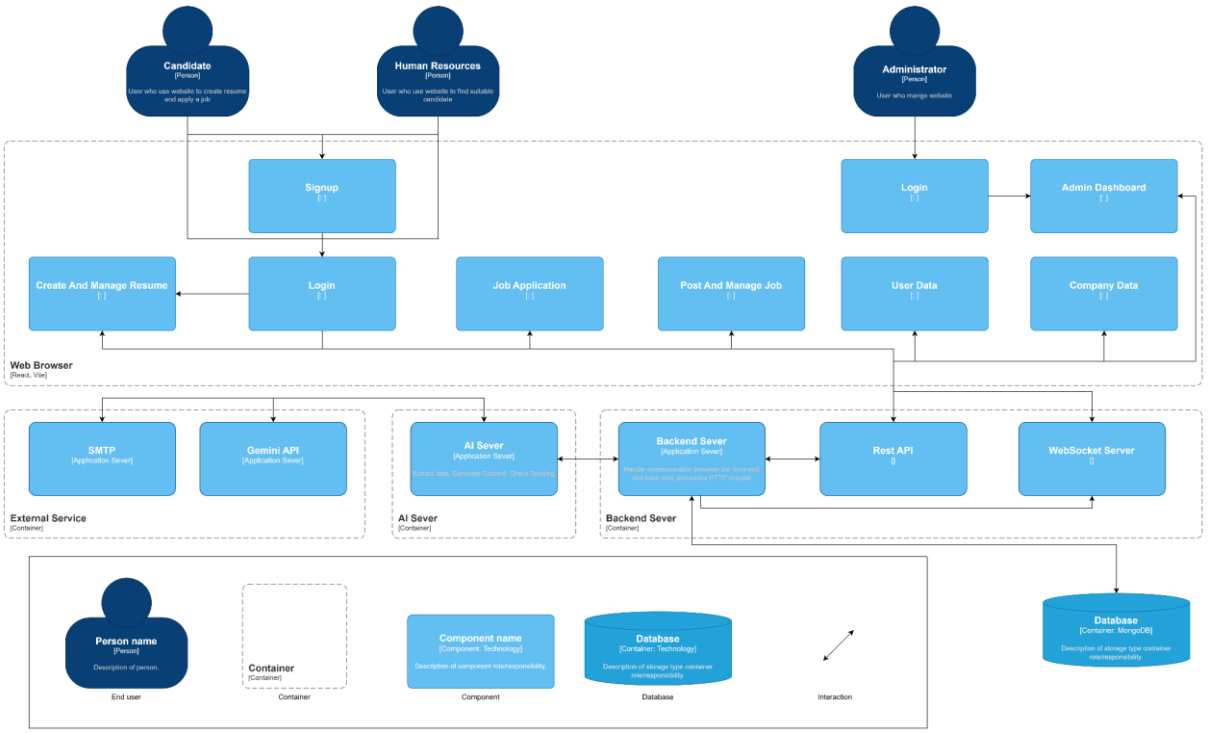

Figure 5.1 *RGA C&C View based on Web Application.*

Prose:

The diagram outlines the architecture of a job application platform, representing its key actors, functionalities, and system components. The system supports three primary user roles: Candidate, Human Resources, and Administrator, each with distinct responsibilities. These users interact through a Web Browser interface with backend components such as application servers, APIs, and databases, ensuring seamless data flow and operation.

The platform incorporates:

- Candidate-focused Features: Account management (Signup, Login), Resume creation and management, and Job application submissions.
- Human Resources Features: Job posting and management, Candidate search and evaluation.
- Administrator Features: Website management through an admin dashboard, Management of user and company data.

System components include a backend server for handling HTTP requests, an AI server for generating and verifying content, and a rest API to facilitate communication. External services like SMTP handle email notifications. The MongoDB Database serves as the primary data storage, ensuring reliable and scalable storage for user, job, and company data.

Table 5.1 *Role And Responsibility of Elements in C&C Figure*

Role	Responsibility
Web Browser	It serves as the user interface for candidates, human resources, and administrators to interact with the system.
Signup	Manages user registration for Candidates and Human Resources.
Login	Authenticates users and ensures secure access to system features.
Create and Manage Resume	Allows Candidates to build, edit, and manage resumes.
WebSocket Services	Handles real-time job application notifications.
Job Application	Enables Candidates to apply for jobs, triggering notifications via WebSocket.
Post and Manage Job	Allows Human Resources to create, update, and manage job postings.
Admin Dashboard	It provides tools for administrators to manage the platform, including users, jobs, and companies.
User Data	Handles CRUD operations for user-related data.
Company Data	Handles CRUD operations for company-related data.

Backend Server	Processes HTTP requests, manages application logic, and interacts with the database and external services.
AI Server	Handles content generation, data extraction, and spell-checking to enhance user input.
WebSocket Server	Enables real-time notifications for relevant events, such as when a Candidate applies for a job.
SMTP	Manages email notifications, such as account verification and password recovery.
Rest API	Serves as an interface for communication between the frontend and backend.
MongoDB Database	Stores and organizes system data, including user accounts, resumes, job postings, and application logs.

5.2. Module view

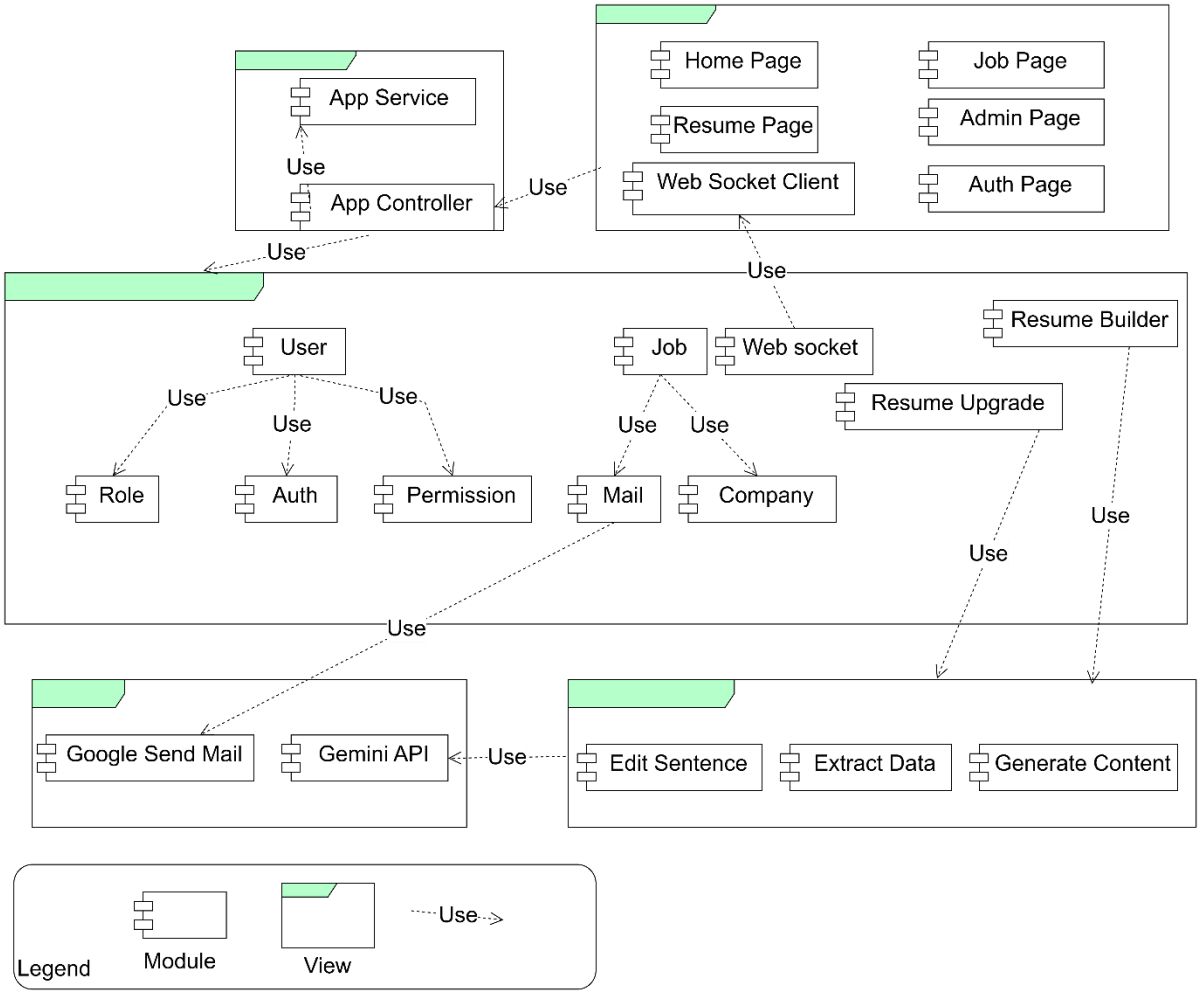

Figure 5.1 *Module View Web Application.*

5.3. Allocation View

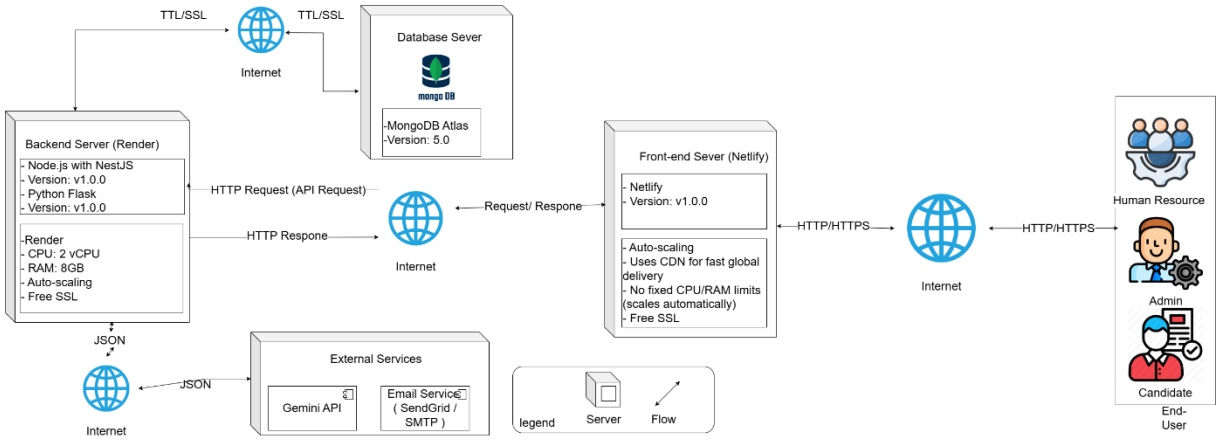

Figure 5.2 *RGA Allocation View.*

Prose:

This allocation view depicts the architecture of the Resume Builder System, showcasing its key components and their interactions. The Web Server is hosted on Netlify and serves as the frontend delivery system.

The backend resides in the Application Server, which handles API requests using NestJS and performs AI-powered text processing with Flask. The Database Server, hosted on MongoDB Atlas, stores structured data such as resumes, user profiles, and job information.

External integrations include the Gemini API for natural language processing tasks (e.g., generating summaries) and an Email Service for sending notifications and alerts. Users—including Admins, HR professionals, and Candidates—interact with the system through secure HTTPS connections, ensuring robust communication between all components.

Role & Responsibilities

Table 5.2 *Role And Responsibility of Elements in Allocation View Figure*

Component	Role	Responsibility
Web Server	**Frontend Delivery**	Delivers the frontend application using Netlify with CDN for fast content distribution.
Application Server	**Backend Processing**	Handles API requests and AI logic using Node.js with NestJS and Python Flask on Render.
Database Server	**Data Store**	Stores structured data on MongoDB Atlas, including user profiles, resumes, and job-related information.
Gemini API	**External Service**	Offers AI-based natural language processing for resume optimization and text generation.
Email Service	**External Service**	Sends notifications and updates to users through SMTP or SendGrid integration.

5.4. Database Design

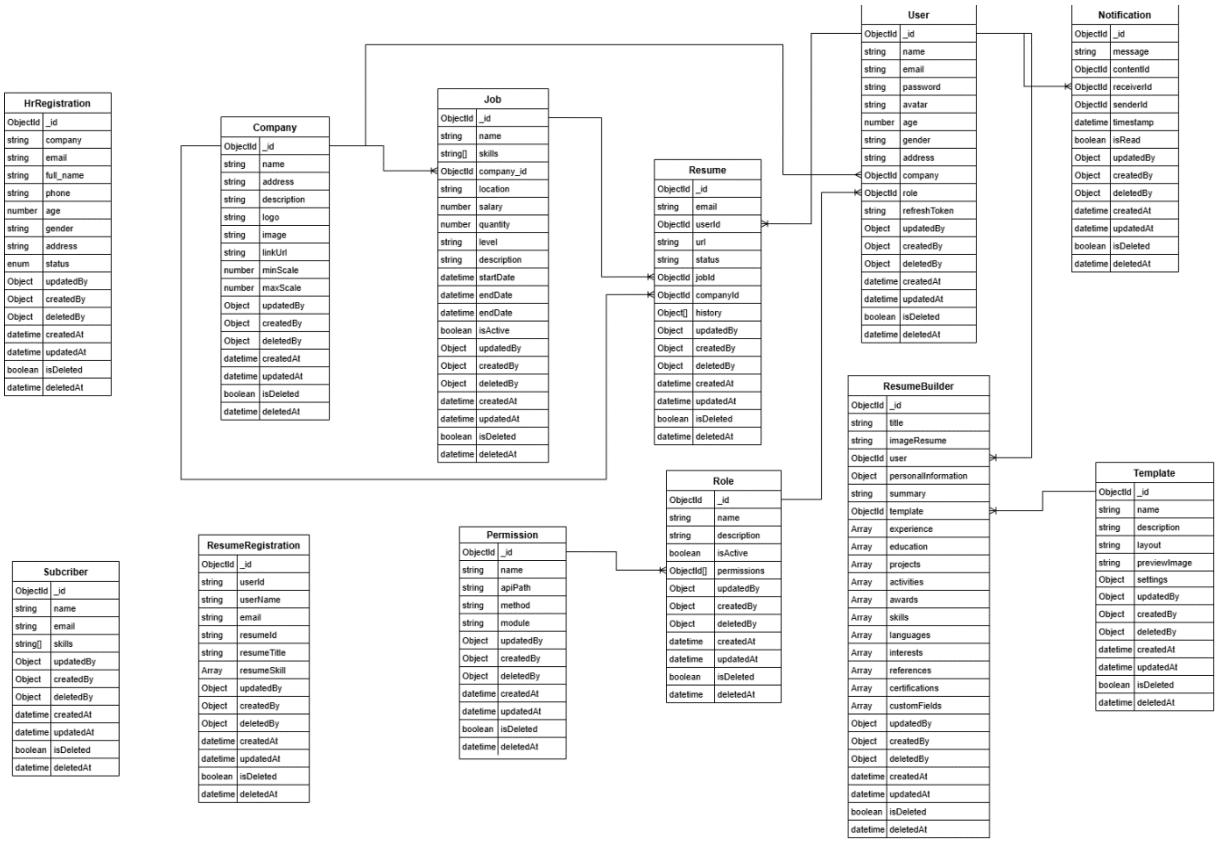

Figure 5.3 *Entity References Diagram.*

6. Artificial Intelligence Model

6.1. Artificial Intelligence Model Training

6.1.1. Artificial Intelligence Predicts The Field Name

Algorithms and techniques used:

The model uses the Random Forest Classifier algorithm - an ensemble machine learning method that combines multiple decision trees to make predictions. Each decision tree in the forest is trained on a subset of the training data and the final model is the result of combining these decision trees.

Reasons to Choose Random Forest:

- Ability to Handle Complex Data: Your data includes instances of text data that are not clearly structured, so Random Forest is a good fit for this situation. While decision trees can work well with numeric features, they can also handle text data well after being vectorized.

- Overfitting Resistance: Random Forest is very resistant to overfitting, especially when you have a large amount of data. The use of bagging and random subspaces helps reduce the influence of inaccurate trees, resulting in a better-generalized model.
- Efficient in Big Data Analysis: When there are many features (like in your case, the words in field_data), Random Forest can handle and make accurate predictions without too much fine-tuning.
- Ease of Deployment: After training the model, Random Forest is very easy to deploy and use for prediction. You can store the model and vectorizer (TF-IDF) and reuse them to classify new instances without retraining.

AI Pipeline:

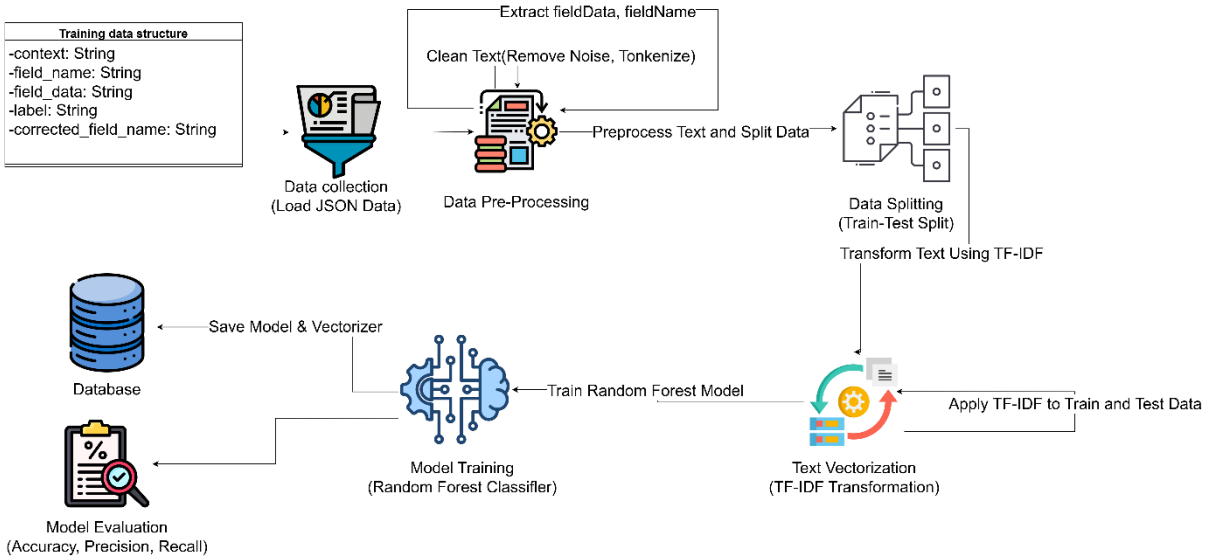

Figure 6.1 *Ai-pipeline Predicts Model Training.*

Step:

1) Read Data: Read data from JSON, and CSV sources, to extract necessary information like field_data, label, and corrected_field_name. prepare it for the next steps.
2) Data Preprocessing: Clean the data and prepare features, and labels for the model, and prepare a list of inputs and target labels (outputs).
3) Split Data Into Training and Testing Sets: Create a training set to train the model and a test set to evaluate the model.
4) Convert Text Data to Numeric Vectors: Purpose: Convert text data (field_data) into a format that machine learning models can understand and process. Use

TfidfVectorizer to convert text into numeric vectors. TfidfVectorizer will automatically clean and remove unimportant words (stopwords) and use statistical algorithms to calculate the frequency of words.

5) Model Training: Train the classification model to learn rules from the training data and predict labels. Use RandomForestClassifier to train the model. Once trained, the model can be used to predict labels for the test set or new data.
6) Save Model and Vectorizer: Save the trained model and vectorizer used so that Joblib can reuse them later without having to retrain them.

Tools and libraries used: Visual Studio Code, Sklearn, Random Forest Classifier, TfidfVectorizer.

Training results:

File: Predict_model.pkl, Vectorizer.pkl

Accuracy: 0.618629353487897

6.1.2. Artificial Intelligence Predicts Field Name Accuracy

Algorithms and techniques used:

This model uses natural language processing (NLP) and deep learning methods to check the match between field data and field names. This is a binary classification problem, where the goal is to determine whether field data matches a field name.

Reasons to Choose NLP, Deep Learning, And LSTM:

Data such as field_name and field_data are natural text. NLP provides techniques such as tokenization, which converts text strings into forms that can be processed by machine learning models, through word segmentation and assigning indexes to words (tokens). This allows the model to understand the structure and semantics of the text.

LSTM is a type of recurrent neural network (RNN) that can store long-term information and overcome the limitations of traditional RNN in learning long-term relationships. For the problem of checking the match between field_name and field_data, LSTM can memorize the characteristics of field names and field data in long strings, helping the model make more accurate decisions about the match.

Each text string (field_name and field_data) can contain long or complex information, and LSTM will help the model learn long-term dependencies between words in the string, such as determining whether a field name like "Phone Number" matches data that is a valid formatted number string.

AI Pipeline:

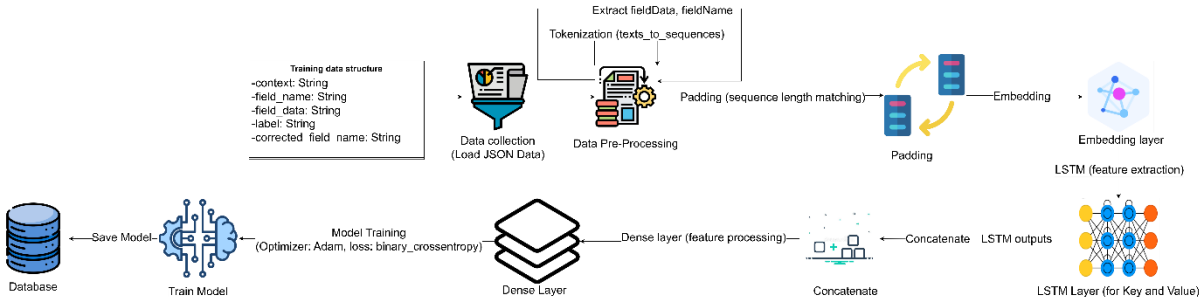

Figure 6.2 *Ai-pipeline Predicts Accuracy Model Training.*

Step:

1) Read Data: Data is loaded from a JSON file with fields field_name, field_data, and label. field_name is the name of the field (e.g. "Email", "Phone Number", "Address"). field_data is the actual value entered for that field (e.g. "john.doe@example.com", "123456789"). label determines whether the field value matches the field name (1 for true and 0 for false).
2) Data Preprocessing: Clean the data and prepare features, and labels for the model, and prepare a list of inputs and target labels (outputs). Text strings (field_name, field_data) are converted to numeric strings (tokens) using a Keras Tokenizer. Each word in the data is assigned a unique index.
3) Padding: The number of strings is re-aligned to the same length (padding) to ensure the model can process them evenly.
4) Embedding Layer: The numeric strings are converted into fixed-length embedding vectors. Using Embedding helps the model understand how words are related to each other in multidimensional space.
5) LSTM Layer: LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that has the ability to remember information for a long time. In this case, LSTM is used to extract features from the text strings of field_name and field_data.

6) Concatenate: The output from the LSTMs for field_name and field_data are concatenated together to combine information from both fields.
7) Dense Layer: A full (Dense) layer of size 64 is used to generate combined features from the two strings.
8) Model Training: The model is trained using the binary_crossentropy loss function (suitable for binary classification) and optimized using the Adam optimizer. The training data is split into two parts: one for training and one for testing (validation split = 0.2).
9) Save the model and tokenizer: After training, the model and tokenizer are saved for later use for testing or real deployment.

Tools and libraries used: Visual Studio Code, Numpy, Tensorflow, Keras, Dense, Embedding, LSTM, Concatenate.

Training results:

File: PredictAccuracy_model.pkl, Vectorizer.pkl

Accuracy: 0.9809

6.2. AI Model Usage

6.2.1. Artificial Intelligence Predicts The Field Name

Loading the Model and Tokenizer:

When the system needs to use the AI model to perform prediction (inference), we will reload the model and tokenizer from the saved files. The code to load the model and tokenizer into the system will be as follows:

```
# Tải mô hình đã lưu
model = joblib.load(model_dir)
# Tải vectorizer (hoặc tokenizer) nếu cần
vectorizer = joblib.load(token_dir)
```

Model Usage Process:

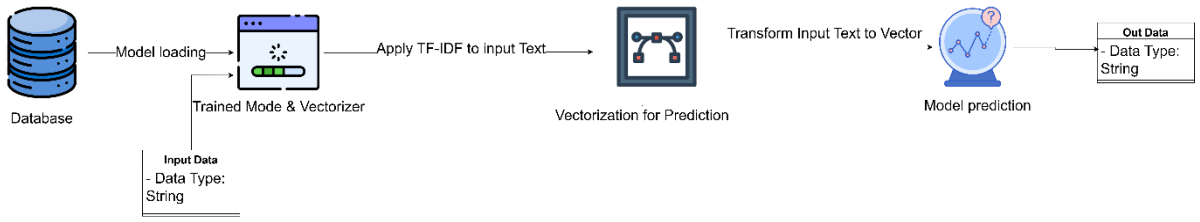

Figure 6.3 *Predicts Model Using.*

Step:

1) Load Pre-trained Model and Vectorizer: Load the trained machine learning model file using the joblib.load method. Load the corresponding vectorizer to transform input text into vectors.
2) Load Input Data: The input should be in text format.
3) Transform Input Text Using TF-IDF: The input text is converted into a numerical format using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer. The transformation creates a vector representation of the text that can be processed by the model.
4) Make Predictions: Pass the vectorized input to the pre-trained model.
5) The model processes the input and predicts the corresponding output.
6) Return Output: The output prediction is returned in the text format.

Tools and libraries used: Visual Studio Code, Joblib, Python OS module.

6.2.2. Artificial Intelligence Predicts Field Name Accuracy

Loading the Model and Tokenizer:

When the system needs to use the AI model to perform prediction (inference), we will reload the model and tokenizer from the saved files. The code to load the model and tokenizer into the system will be as follows:

```
# Tải mô hình đã huấn luyện từ file .keras
model = tf.keras.models.load_model('final/final_model.keras)
# Tải tokenizer từ file JSON
with open('final/final_tokenizer.json', 'r') as f:
    tokenizer_json = f.read()
tokenizer = tokenizer_from_json(tokenizer_json)
```

Model Usage Process:

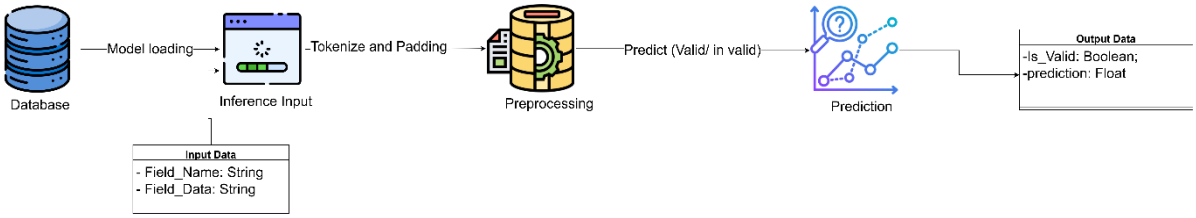

Figure 6.4 *Predicts Accuracy Model Using*

Step:

1) Load Pre-trained Model and Vectorizer: The model is loaded using tf.keras.models.load_model. The tokenizer is loaded and reconstructed from JSON using tokenizer_from_json.
2) Load Input Data: Input data includes: Field Name: A string representing the key or field (e.g., "email"). Field Data: A value or object associated with the field (e.g., "example@example.com" or a nested dictionary). Input data is preprocessed using a custom function, convert_to_string, to ensure it is in string format suitable for tokenization.
3) Preprocessing: The field_name and field_data are tokenized into sequences using the loaded tokenizer. The tokenized sequences are padded to match the expected input shapes of the model. Padding is applied using pad_sequences with the appropriate maxlen values for both field_name and field_data.
4) Prediction: The padded sequences for field_name and field_data are passed as inputs to the model. The model produces a prediction value. If prediction > 0.5, the field_name is considered valid. Otherwise, it is considered invalid.
5) Generate Output: The prediction result is returned as a structured dictionary. The output includes is_valid: A boolean indicating the validity of the field_name. prediction: The model's raw prediction score.

Tools and libraries used: Visual Studio Code, Tensorflow, Keras.

6.2.3. NLP Model

Overview:

The spaCy pipeline is designed to process input text (Text) and transform it into a Doc object containing linguistically analyzed information. Each step in the pipeline performs a specific task, such as tokenization, part-of-speech tagging, or named entity recognition, enabling seamless text analysis.

Why Use spaCy:

spaCy is an essential tool for natural language processing (NLP) because it provides an efficient, scalable, and easy-to-use pipeline for text analysis. The modular architecture allows for flexible customization, enabling users to add or remove components depending on their use case.

Its capabilities, such as fast tokenization, accurate POS tagging, dependency parsing, and named entity recognition, make it suitable for a wide range of tasks, from information extraction and text classification to building chatbots and search engines. Additionally, spaCy's pre-trained models are optimized for performance, ensuring high accuracy while processing large datasets quickly. By transforming raw text into a structured Doc object, spaCy empowers developers and researchers to extract meaningful insights with minimal effort.

AI Pipline:

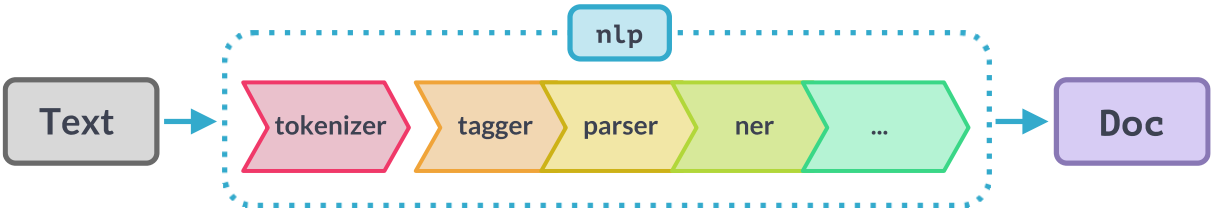

Figure 6.5: *NLP Model*

Step:

1) **Text**: The input text data, which can be a sentence, paragraph, or document.
2) **Tokenizer**: Splits the input text into tokens (words or symbols).Example: "I love spaCy." → ["I", "love", "spaCy", "."].
3) **Tagger**: Assigns part-of-speech (POS) tags to each token. Example: "love" is tagged as a verb (VERB).
4) **Parser**: Analyzes the syntax of the text and builds a dependency tree for the sentence. Identifies relationships between words, such as subject and predicate.
5) **NER (Named Entity Recognizer)**: Detects named entities in the text, such as people, places, or organizations. Example: "spaCy" is recognized as ORG (organization).
6) **(Other Components)**: The pipeline can be customized with additional components, such as text categorization or user-defined modules.

7) **Doc:** The final output is a Doc object that contains all the analyzed linguistic information, including tokens, POS tags, dependency parses, and named entities

spaCy is an essential tool for natural language processing (NLP) because it provides an efficient, scalable, and easy-to-use pipeline for text analysis. The modular architecture allows for flexible customization, enabling users to add or remove components depending on their use case.

Its capabilities, such as fast tokenization, accurate POS tagging, dependency parsing, and named entity recognition, make it suitable for a wide range of tasks, from information extraction and text classification to building chatbots and search engines. Additionally, spaCy's pre-trained models are optimized for performance, ensuring high accuracy while processing large datasets quickly. By transforming raw text into a structured Doc object, spaCy empowers developers and researchers to extract meaningful insights with minimal effort.

7. References

- *https://www.docker.com*
- *https://nextjs.org/*
- *https://fastapi.tiangolo.com*

8. Attachment

https://drive.google.com/drive/folders/1RZtmGyhkQ0eHeKxCpVgUtui4NJLl_eii?usp=sharing